International Journal of Computer Information Systems and Industrial Management Applications
ISSN 2150-7988 Volume 18(1s) 2026 pp. 1-22

https://doi.org/10.70917/ijcisim-2026-2148
Article

A New Metaheuristic-Based Load Balancing Framework Using Grey Wolf Optimization And Centroid Opposition-Based Learning

P. Rajesh[1*], K. Maharajan[2], Jayalakshmi Murugan[3]

[1]Department of Computer Science and Engineering Kalasalingam Academy of Research and Education, Krishnankoil, Tamilnadu, India
[2]Department of Computer Science and Engineering Kalasalingam Academy of Research and Education, Krishnankoil, Tamilnadu, India E-mail:
[3]Department of Computer Science and Engineering-Cyber Security, RAMCO Institute of Technology, India
E-mail: [1*]rajeshmephd30@gmail.com, [2*]maharajank@gmail.com, [3*]jayalacsmi@gmail.com

Abstract: One of the key challenges in cloud computing is the efficient allocation of load among the resources, which are distributed dynamically and may contain resources of different capacities in the form of virtual machines (VMs). This paper presents an improved metaheuristic, which is the Grey Wolf Optimiztion (GWO) algorithm combined with Centroid Opposition-Based Learning (COBL) algorithm for a novel load balancing scheme. COBL facilitates the convergence process to help improve the exploration/exploitation ratio of GWO by producing competitive opposite solutions around the centroid of the population. The COBL-GWO algorithm is proposed for task scheduling in cloud computing to enhance the main performance metrics such as make span, resource utilization, execution time and memory usage. The experimental tests with different task sizes and VM configurations (5 and 10) show that COBL-GWO is clearly superior to the standard GWO, IGWO, CPSO, PSO and GA approaches. Specifically, COBL-GWO shows good performance in terms of computing cost, resource utilization, and make span, and is thus a strong and scalable solution to efficiently schedule and balance tasks in cloud computing.

Keywords: Cloud computing; Swarm intellignce; Grey wolf optimization; Centroid-opposition based learning; population diversity;

1. INTRODUCTION

Cloud Computing has revolutionized the way computing services and resources are made available and used. Cloud computing offers unprecedented flexibility, scalability and cost efficiency by enabling access to a shared pool of reconfigurable computing resources like servers, storage, apps and services [1]. As more and more companies and organizations shift their workloads to cloud platforms, the management of the computational resources becomes a challenge. A crucial concern in this field is load balancing, which has a direct impact on cloud-based services' cost, dependability, and performance. Distributing workloads and computing activities evenly among several resources, like servers, clusters, and virtual machines (VMs), is known as load balancing. If load balancing is implemented correctly, no resource will be overloaded, and other resources remain unused. Load balancing is particularly crucial in the environment of cloud computing, due to the dynamism and diversity of cloud infrastructures. A system can have poor balance that can result in excessive response time, increased execution costs, waste of resources, and poor user experience. In addition, it may cause Service Level Agreement (SLA) breach and instances of poor application performance in mission-critical applications.

Despite their simplicity, traditional load balancing algorithms such as Round Robin, Least Connections, and Min-Min often fail to perform well in complex cloud-based systems with dynamic resource capacities and workloads. As a result of these deterministic or static algorithms, they are not the most appropriate for large-scale

and dynamic workloads because they cannot adapt well to the changes of tasks and systems in real time as required. Therefore, to guarantee effective resource use and enhanced system performance, sophisticated and flexible load balancing mechanisms are needed. Recently, Metaheuristic algorithms have received significant attention due to their applications on tackling complex problems in cloud computing, including load balancing and task scheduling problems. The algorithms consist of those that are inspired by biological evolution, natural occurrences or animal behavior, and which can escape local optima to reach near-optimal solutions in wide search spaces. Ant Colony Optimization (ACO), Particle Swarm Optimization (PSO), Firefly Algorithm (FA), and Genetic Algorithm (GA) are a few examples.

GWO has proven to be a promising approach among these. The successful global optimization of GWO has been demonstrated by its ability to balance between the exploitation (local search) and exploration (global search) processes, which is inspired by the hunting and leadership style of the gray wolves in nature. The system is based on the gray wolf hunting process where alpha, beta, and delta wolves suggest good solutions and guide the search process of other agents (omega wolves) in the population. For a few problem cases, even though the GWO is successful, there still remain limited exploration and early convergence in some problem cases. In practice, cloud scheduling problems often contain high-dimensional, dynamic or nonlinear problems, and may lead to poor performance. Improvements to GWO that can broaden the search space and steer clear of local optima traps are required due to this constraint.

COBL is one approach that has proven to be a useful technique in enhancing the diversity and convergence rate of the metaheuristic algorithms. At the heart of OBL is the process of optimisation – simultaneously considering both an estimate, and its counter-proposal. In this way, the algorithm can find a better (or close to an optimum) solution more quickly and search a wider area of the solution space. These two opposite solutions are generated with respect to the centroid of the existing population, not the limits of the search space as in COBL, an improved version of OBL. This way, you get a more knowledgeable and flexible search tool. COBL can be integrated with GWO without affecting the exploitation capacity of the original algorithm, while achieving an improvement in global exploration ability. In this suggested study, we develop a hybrid algorithm named COBL-GWO which combines COBL with the conventional GWO algorithm. The primary goal is to optimize execution time, memory consumption, reduce make span and maximize resource utilization, for better job scheduling and load balancing in cloud computing systems. The highlights of this work are:

- Proposed a novel hybrid load balancing scheme by combining GWO and COBL in cloud computing environment.
- Efficient scheduling and resource allocation in dynamic and heterogeneous cloud infrastructures.
- Improved the exploration and exploitation capability in the optimization process using population initialization and updating mechanism based on COBL.
- Avoids premature convergence and prevents the occurrence of local optimum problems that often occurs in standard optimization algorithms.
- A complete performance assessment with different tasks sizes and two VM configurations (5 VMs, 10 VMs).
- Detailed performance metrics analysis such as make span, utilization of resources, execution time and memory utilization.
- Significant improvement of COBL-GWO in the load balancing task compared with other algorithms.

The remaining of the paper is organized as follows : Section 2 discussed related work. Section 3 is discussed about the problem statement. Section 4

discussed research methods including GWO, COBL and proposed COBL-GWO. The experimental results are discussed in section 5. In the last section 6, limitations and possible future research are presented.

2. RELATED WORKS

Load balancing is one of the essential operations in cloud computing, which plays an important role in distributing the workloads properly across the available computational resources for achieving the objective of high availability, stability of the system, and optimum utilization of resources. Recently, a number of studies have been conducted to explore the different metaheuristic and hybrid optimization techniques to improve load balancing in

cloud computing systems. The article by J. Zhou et al. (2023) [2] is a comparative study of some metaheuristic load balancing algorithms in terms of such metrics as makespan, response time, and resource utilization.

Their results indicated that PSO can be effectively used in terms of many performance indicators. But the researchers primarily address comparative analysis and do not develop an innovative method of optimization. A multi-objective load balancing model called MCGEO, was proposed by K. V. Kumar et al. (2023) [3], by combining both the GEO and CMBO methods. The model enhances convergence and high throughput is achieved. However, the task of hybridization adds to the computational load, and can also have implications on scalability in large-scale cloud architectures. M. Sumathi et al. (2026) [4] suggested the HGAPSO algorithm as a hybridization of GA, PSO, and HHO, and has shown significant reduction in response time, makespan, and computational cost. Although it has shown very good performance, the combination of several algorithms adds complexity to the model and complicates the process of tuning the parameters.

M. S. R. Krishna (2025) [5] proposed a hybrid optimization model (Meta-RHDC) based on deep learning and optimization to optimize the load balancing in a dynamic system. The strategy is very effective in terms of scalability, and resource use. Nonetheless, reinforcement learning models are very time consuming and resource consuming. M. S. Al Reshan et al. (2023) [6] have suggested a hybrid GWO-PSO algorithm to enhance the speed of convergence and the ability to optimize globally. The approach decreases the response time, but it can still experience premature convergence in highly dynamic environments. The hybrid model proposed by G. Lokesh et al. (2025) [7] is a combination of Modified Dragonfly Algorithm and ABi-LSTM to predict resource states and optimize the process of resource scheduling. Though the model has high throughput, it relies heavily on deep learning, making system complexity and training more complex. The article by S. Singhal et al. (2024) [8] suggested a load balancing algorithm based on the Rock Hyrax system, which was oriented at the energy consumption and the quality of services as the key parameters. The approach minimises makespan and energy use, although its behaviour in large scale real-time conditions is unclear.

Hybrid DCCO algorithm (CMBO + DXO) has been developed by K. Geeta et al. (2023) [9] to optimize server load and resource utilization. Although the technique demonstrates better performance, it is not flexible to dynamic changes in workload. G. Long et al. (2025) [10] made a proposal of a PSO-Fuzzy-based scheduling to optimize the execution of tasks and energy consumption. Fuzzy logic integration helps in decision-making, although the integration raises the complexity of system design. The article by S. Ghafir et al. (2024) [11] proposed a multi-objective PSO with Double Deep Q-learning model and feedback controller based on GANs to resolve SLA violations. Despite the approach enhancing fault tolerance, it is associated with high computational complexity and has to be extensively trained. P. S. Priya et al. (2025) [12] suggested the PTBO algorithm in the task scheduling, which would lead to a decrease in the makespan and energy used. Nevertheless, the performance of the algorithm has largely been tested on a simulated setting, which restricts applicability to the real world. M. Sumathi et al. (2023) [13] proposed a hybrid HGO-ACO model to load balance, enhance execution time and response time. Although efficient, hybrid models can often be sensitive to parameter tuning. P. J. Assudani et al. (2023) [14] suggested an IPSO-GA hybrid model of resource scheduling efficiency. The model enhances makespan and resource usage, however, hybridization adds computational overhead. Rock Hyrax optimization was applied in S. Singhal et al. (2023) [15] to achieve the best energy efficiency and QoS. Although there are improvements being realized, the improvements are quite moderate in relation to highly sophisticated hybrid models.

M. Jesi et al. (2024) [16] proposed the MFO-LB methodology which is based on the optimization of mayflies, and resulted in a reduction in response time and makespan. Nevertheless, the algorithm can have problems with the rate of convergence in complicated situations. SCEHO-IPSO was proposed by K. J. Rajashekar et al. (2023) [17] to enhance the exploration and exploitation capabilities. The algorithm does not suffer the problem of local optima, but it positively increases the complexity of the algorithm. The algorithm of EDLB, created by R. Zhanuzak et al. (2024) [18] helps to minimize the infringement of SLA by actively distributing the resources. Although it can greatly enhance makespan and resource utilization, it is very much dependent on proper system state prediction. V. Hayyolalam et al. (2025) [19] proposed CBWO, combining chaos theory with Black Widow Optimization, with significant improvements in makespan and energy consumption. Nonetheless, chaotic models can be used to bring about instabilities under specific circumstances.

The approach of the dynamic load balancing presented in T. A. Murshedi et al. (2024) [20] is based on the approach of the resource allocation matrix. Though it is more economical with resources, it does not have high levels of optimization. K. Ramya et al. (2023) [21] suggested HDWOA-LBM (a combination of Dingo Optimization

and Whale Optimization) that can enhance reliability and throughput. Nevertheless, it is hard to tie the concept of exploration and exploitation. F. Kaplan et al. (2025) [22] tested GA-PSO hybrid scheduling in normal and DDoS conditions, showing better resilience. However, the model is not flexible to the quickly evolving workloads. A combination of ACO and Tabu search, ACOTS was introduced by J. P. Gabhane et al. (2023) [23] and allows fast data processing. Nonetheless, the strategy might experience challenges related to scalability in large clouds. The U. K. Lilhore et al. (2025) [24] suggested a hybrid WWO-ACO model that enhances the efficiency and consumption of energy. Although it is effective, hybridization introduces complexity in computations. The HKHW-DBNM created by P. Neelakantan et al. (2023)

[25] was created to serve VM load balancing, which optimizes resource use and execution time. Nevertheless, it does not have a dynamic adaptability. With workload management, K. Nivitha et al. (2025) [26] introduced CBBM-WARMS using clustering and fuzzy logic. Although it minimizes SLA violations, it consumes a lot of computational resources.

To enhance the distribution of loads and the time of execution, a modified version of Min-Min heuristic was offered by A. Choudhary et al. (2024) [27]. Despite being effective, heuristic techniques might not work well in dynamically changing situations. L. Du et al. (2024) [28] introduced a dynamic load balancing scheme that adjusts to the changing workloads. The strategy enhances throughput but does not have the advantages of hybrid optimization. P. Geetha et al.(2024) [29] proposed the use of IC&BA to optimize multiple goals with the combination of BES and CA algorithms. Nevertheless, scalability is impacted by an increase in the complexity of the model. TSO-MCR was recommended by A. Boroumand et al. (2025) [30] as a means of multi-objective scheduling to enhance reliability and cost efficiency. Yet, the approach requires careful tuning of multiple objectives. A hybrid model comprising of clustering, round-robin, and GA to solve the task scheduling problem was developed by N. Elsakaan et al. (2024) [31]. Although it improves the performance and scalability, the multi-layer architecture increases the computational overhead.

Table 1 : Comparative analysis of load balancing approaches

Ref. No	Methods	Key Contributions	Merits	Demerits
[2]	PSO (Comparative Study)	Evaluated metaheuristic algorithms using QoS metrics	Good performance in makespan & response time	No novel method proposed
[3]	MCGEO (GEO + CMBO)	Multi-objective load balancing with improved convergence	High throughput, better optimization	High computational complexity
[4]	HGAPSO (GA+PSO+HHO)	Hybrid model improving APT, makespan & cost	Significant performance improvement	Complex parameter tuning
[5]	Meta-RHDC (RL + DL + Optimization)	Dynamic load balancing using RL & deep learning	High scalability, better utilization	High training cost
[6]	GWO-PSO Hybrid	Improved convergence and global search	Reduced response time	Risk of premature convergence
[7]	Dragonfly + ABi-LSTM	Predictive load balancing using DL	High throughput	Increased system complexity
[8]	Rock Hyrax Algorithm	QoS-aware energy-efficient	Reduced makespan & energy	Limited real-time

		scheduling		validation
[9]	DCCO (CMBO + DXO)	Hybrid optimization for server load reduction	Improved resource utilization	Less adaptive to dynamic loads
[10]	PSO-Fuzzy	Fuzzy-based VM scheduling	Better decision-making	High design complexity
[11]	PSO + DDQ + GAN	SLA-aware intelligent scheduling	High fault tolerance	Very high computational cost
[12]	PTBO	Artist-inspired optimization for scheduling	Reduced makespan & energy	Limited real-world validation
[13]	HHO + ACO	Hybrid model for improved execution time	Better response time	Parameter tuning required
[14]	IPSO-GA	Improved scheduling & resource allocation	Efficient resource utilization	Increased overhead
[15]	Rock Hyrax	QoS-based optimization	Energy efficient	Moderate improvements only
[16]	MFO-LB (Mayfly)	Load balancing using swarm intelligence	Reduced RT & makespan	Slow convergence in complex cases
[17]	SCEHO-IPSO	Improved exploration & exploitation	Avoids local optima	High complexity
[18]	EDLB	SLA-aware proactive scheduling	Improved resource utilization	Depends on prediction accuracy
[19]	CBWO (Chaos + BWO)	Energy-efficient hybrid optimization	Significant performance gain	Possible instability
[20]	Resource Matrix Model	Dynamic resource allocation	Improved efficiency	Lacks advanced optimization
[21]	HDWOA-LBM	Hybrid DOA + WOA optimization	Improved reliability & throughput	Balancing issues
[22]	GA-PSO	Resilient scheduling under DDoS	Improved robustness	Limited adaptability
[23]	ACOTS (ACO + TS)	Multi-resource load balancing	Faster processing	Scalability issues
[24]	WWO-ACO	Hybrid scheduling optimization	Reduced cost & energy	High computational

				cost
[25]	HKHW-DBNM	VM load balancing method	Improved utilization	Not adaptive
[26]	CBBM-WARMS	Clustering + fuzzy scheduling	Reduced SLA violation	High complexity
[27]	Modified Min-Min	Improved heuristic scheduling	Reduced completion time	Not suitable for dynamic systems
[28]	Dynamic Metaheuristic	Adaptive load distribution	Improved throughput	No hybrid optimization
[29]	IC&BA (BES + CA)	Multi-objective load balancing	Better optimization	Scalability issues
[30]	TSO-MCR	Multi-objective scheduling	Improved cost & reliability	Complex tuning
[31]	Hybrid (Clustering + GA + RR)	Multi-layer scheduling framework	High scalability	High overhead

The current hybrid optimization approaches have the disadvantage of being highly computationally complex, not allowing them to be deployed in real-time applications. The majority of the approaches do not include adaptive hyperparameter tuning and therefore, they do not perform optimally in terms of their response to dynamic workloads. Further, the problem of premature convergence and local optima have not been solved in metaheuristic algorithms. The energy use and the complexity of the model, particularly in deep learning-based methods. The absence of the real-world validation, where most of the studies have been carried out solely using simulation tools.

3. PROBLEM FORMULATIONS

It is not a straightforward problem to map all the tasks to available VM in cloud computing scenario, and to find an optimal solution. The challenge is to come up with a productive task scheduling algorithm for the sake of this case, which should keep the load of VM and allocate all of the user's task to right resource. This paper depicts the direct significance of the framework structure, tasks and the resources. The essential objective of proposed technique called IGWO to optimally schedule all incoming task to the available VMs so that it can reduce makespan and increase machine utilization in cloud computing, each task must be allocated to only one VM. Suppose a cloud data center contains of x number tasks ($T = \{T_1, T_2, \dots, T_X\}$) and y number of heterogeneous VM ($VM = \{VM_1, VM_2, \dots, VM_X\}$. But the condition for execution of such tasks is $x > y$. All tasks are of length (T_1) and communicated as MI (million instruction). The execution speed of y^{th} VM is displayed as million instructions per second (MIPs) and in order to get our goal, we calculate the service rate of VM by following equation,

$$\sum_{j-1}^{v} VM_{sr} = \sum_{j-1}^{v} VM_{mips} \times \sum_{j-1}^{v} VM_{cpu} \qquad (1)$$

The expected execution time (EET) of task perform on VM_j where $j = \{1,2, \dots, y\}$ can be denoted as a following equation,

$$EET_{i,j} = \sum_{i=1}^{x} T_1 \times \sum_{j=1}^{y} VM_{sr} \qquad (2)$$

A single VM is comprises one or more than one tasks. If the number of tasks in VM is more, then total

file size and dependency of tasks which is shown in following equation,

$$T_{\text{totalfilesize}} = T_{\text{individualfilesize}} \times T_{\text{dependency}} \quad (3)$$

When the task is moved from one VM to other then there is probability to take some transfer time. To transfer an individual file size depends on the task transfer speed $TT_{\text{S}}(i, j)$ of VM. The speed of the task transfer is measured as follows:

$$TT_{\text{s}} = \sum_{\text{i=1}}^{\text{x}} T_{\text{individualfilesize}} / \sum_{\text{j=1}}^{\text{y}} VM_{\text{bw}} \quad (4)$$

Task transfer time is the time required to transfer the size of the task file between virtual machine. Therefore, task transfer time $TT_{\text{t}}(i, j)$ represents the time of task i transmit to the VM j and communication time of task i on machine time j is calculated as follows,

$$CT_{\text{i,j}} = T_{\text{totalfilesize}} / TT_{\text{t}} \quad (5)$$

Finally, all tasks' total execution time at any $VM_{\text{j}}$is the sum of task execution time and task transfer time, which is shown in following equation, respectively.

$$ET_{\text{i}} = EET_{\text{i}} + TT_{\text{S}} + CT_{\text{i,j}} + DV_{\text{i,j}} \quad (6)$$

Where $DV_{\text{i,j}}$ is the binary decision variable,

$$DV_{\text{x,y}} = \begin{cases} 1, & if\ T_{\text{i}}\ is\ allocated\ to VM_{\text{j}} \\ 0 & otherwise \end{cases} \quad (7)$$

Makespan (MS) time can be falling when a greater number of tasks are performed successfully at datacenter, it also gives an impact of system throughput.

The MS is the maximum of ET_{i} as computed along eqn., and can be used to determine the total finishing time of all the tasks.

$$MS = ma\{ET_{\text{i}}\} \quad (8)$$

Next optimization goal is to rise the resource utilization (R_{u}) which is considered by the total execution time of all task divided by highest execution time of makespan. Following equation displays the average utilization of the resources,

$$R_{\text{u}} - \frac{\text{ET}_{\text{i}}}{\text{MS}} \quad (9)$$

$$R_{\text{avgu}} - \frac{\sum_{\text{j=1}}^{\text{y}} \text{“u}}{\text{y}} \quad (10)$$

The fitness function (F) is defined as following equation which means that each wolf will be given a better position with optimal solution. WT ($waiting\ time$) is the time for wait the task to allocate on diverse VM.

$$F = WT + MS + R_{\text{u}} \quad (11)$$

Assume that we have three VMs and the following six tasks scheduled as depicted. All tasks are arbitrarily assigned to machines additionally shows the time that each VM takes to execute the tasks. According to MS and R_{u}, we compute makespan and resource utilization of both tables. It is not taken here since all machines are not overloaded and decision variable equals to 1. In table the makespan is 15 s because it is the highest execution time. The utilization of VM1, VM2 and VM3 is 0.87,0.97 and 1 respectively. The utilization of VM1, VM2

Algorithm 1: GWO algorithm

Step 1: Initialize the random populations Step 2: Evaluate the fitness values

Step 3: Identify the best solutions of X_α, X_β and X_δ
Step 4: For each candidate

Step 4.1: update the control parameters and calculate co-efficient
Step 4.2: Calculate the distance from X_α, X_β and X_δ
Step 4.3: Update position of each candidate
Step 4.4: Select the best original and opposite solution Step 4.5: Re-evaluate fitness values
Step 4.6: Update X_α, X_β and X_δ
Step 4.7: Repeat the steps until meet the convergence Step 5: End for
Step 6: Optimal solutions

and VM3 is 1,1 and 0.92 respectively for the case of table makespan is 12 s. It is usually assumed that all tasks are assigned to VMs and minimize the makespan and maximize the usage.

a) VM availability

When number of tasks (x) is assigned to number of resources (y) then numbers of resources are used which is defined as follows,

$$VM_{\text{use}} = \sum_{i=1}^{x} T_i \qquad (12)$$

$$VM_{\text{use}} = \sum_{i=1}^{x} T_i \qquad (12)$$

$$VM_{\text{use}} = \sum_{i=1}^{x} T_i \qquad (12)$$

b) VM selections

Task is being allocated to a VM, depending on the arrival time of the task. This implies that the tasks with the earliest arrival time are allocated to the resources first and the task with the smallest length will be completed first. VM is selected to execute this task if it depends on the length of the next task, workload of VM and capacity of VM, which is calculated by using the following formula: VM capacity is determined by its available processing speed, memory, CPU and bandwidth

$$R_s = \frac{T_{i1} * VML_i}{VM_{sr}} \qquad (13)$$

All time scheduler would like to schedule all tasks to all available VM; depends on the load of VM at the specific time. So, we calculated the load of VM (VML_j)and capacity of VM as follows,

$$VML_i = \frac{T_i \times \sum_{i=1}^{x} T_1}{VM_{sr}} \qquad (14)$$

$$VM_c = VM_{sr} + VM_{bw} \qquad (15)$$

c) VM state

Three states of VM are overload, underload and balanced state. A VM is under loaded it its usage is under 25% of its ability and VM is overloaded if its usage is over 80% of its ability. Else slays is balanced as it shows in next eqn. tasks are transferred from over burden VM to understand VM until all the tasks are balanced. In this case, the tasks are shifted from an over loaded VM to under loaded VM which is defined as follows,

$$VM = \begin{cases} R_u < |VM_c \times 25\%|, & underload \\ R_u = VM_c, & balanced \\ R_u < |VM_c \times 80\%|, & overload \end{cases} \quad (16)$$

4. RESEARCH METHODS

4.1 GWO

Recently, a meta-heuristic optimization method, called the GWO, was proposed for the solution of optimization problems based on swarm intelligence (SI) [32]. This algorithm is a referencing of the social leadership and the hunting behavior of grey wolves in nature, grey wolves are in the top of the food chain and they prefer to live in a pack with a specific interest in this, they have a very strict dominant hierarchy. In this algorithm, the population is divided into four groups: alpha (α),beta (β),delta (δ), and omega (ω). The hunting action in GWO is controlled by α,, and δ. The ω wolves follow these three wolves. In each cycle, the top three wolves, which are called as α,β, and δ are selected and the remaining wolves are called ω and they are forced to circle the top three wolves to find better solutions. During the optimization, in order to mathematically model the encircling behavior, the following equations are as follows:

$$(t + 1) = X_p(t) - A.D \quad (17)$$

$$D = |C.X_p(t) - X(t)| \quad (18)$$

Where t specifies the present iteration, A and C are coefficient vectors, X_p represent the position vector. X specifies the position vector of a grey wolf. The vectors A and C are considered as follows:

$$A = 2a.r_1 - a, \quad (19)$$

$$C = 2.r_2 \quad (20)$$

a are linearly decreased from 2 to 0 over the course of iterations and r_1, r_2 are random vectors in [0,1]. To mathematically simulate the hunting behavior of grey wolves, we assume that α, β, and δ have some greater knowledge about the potential location of prey. Therefore, the first three best solutions found so far are saved and the other search agents (including the) are forced to adjust themselves in accordance to the position of the best search agent. The following formulas are proposed in this regard.

$$D_\alpha = |C_1.X_\alpha - X| \quad (21)$$

$$D_\beta = |C_2.X_\beta - X| \quad (22)$$

$$D_\delta = |C_3.X_\delta - X| \quad (23)$$

Where, X_α shows the location of the δ. C_1, C_2 and C_3 are random vectors. X shows the position of the present solution. The above equations are used to compute the estimated distance between the present solution and α,, and δ respectively. The final position of the present solution is considered as follows:

$$X_1 = X_\alpha - A_1.(D_\alpha) \quad (24)$$

$$X_2 = X_\beta - A_2.(D_\beta) \quad (25)$$

$$X_3 = X_\delta - A_3.(D_\delta) \quad (26)$$

$$X(t + 1) = \frac{X_1 + X_2 + X_3}{3} \quad (27)$$

Where, X_α represent the position of the α. X_β shows the position of the β. A_1, A_2 and A_3 are random vectors. t specifies the number of iterations.

4.2 COBL

The main idea of opposition was brought to the field of machine learning in 2005. Since then, several OBL schemes have been successfully applied in metaheuristic algorithms to improve the performance. A detail review on the use and evolution of OBL approaches cab be found in[33]. Let x be a real number which has been determined in the interval $[a, b]$. The opposite number is such that,

$$x^{\vee} = a + b - x \tag{27}$$

A point $X = (x_1, x_2 \dots x_D)$ is given in the D dimensional space, x_i is located in the interval $[a_i, b_i]$. There is a unique opposite point $x^{\vee}{}_\iota$ defined as follows:

$$\widehat{x_1} - a_1 + b_1 - x_1 \tag{29}$$

COBL calculate the centroid opposite the points considering the whole population. Let $X = (X_1 \dots X_N)$ be N points in a D dimensional search space. The centroid of the body can be defined as:

$$M = \frac{x_1 + x_2 + x_3 + \cdots + x_N}{N} \tag{30}$$

The centroid point can be considered as follows:

$$M_j = \frac{1}{N} \sum_{i=1}^{N} x_i, j \tag{31}$$

The opposite-point $\breve{X}$ of a point X of the body is defined as M which is calculated as follows,

$$\breve{X}_\iota = 2 \times M - X_i \tag{32}$$

$$\breve{X}_\iota = 2 \times M - X_i \tag{32}$$

$$\breve{X}_\iota = 2 \times M - X_i \tag{32}$$

The centroid approach can be employed to optimization problems, usually achieving better performance.

5. PROPOSED COBL-GWO

The proposed work combines the concepts of the GWO and COBL to improve the effectiveness of load balancing in cloud computing systems. GWO is an inspired metaheuristic algorithm, which imitates the hunting mechanism and the hierarchy of leadership of the grey wolves. It offers good exploration and exploitation capacity to solve optimization problems like schedule of tasks and allocation of resources. Nevertheless, conventional GWO can have a slow convergence rate and local optima. In an effort to address these shortcomings, COBL is included with GWO. COBL enhances the search process by creating opposite solutions depending on the centroid of the population that increases the diversity of the population and speeds up convergence. This assists the algorithm to get out of local optima, and explore the search space better. In the suggested COBL-GWO model, the initial population of candidate solutions (task-to-VM

Algorithm 2: COBL algorithm

Step 1: Initialize the populations of candidate solutions
Step 2: Compute the centroid of the populations

Step 3: Generate the opposite solution for COBL
Step 4: Evaluate the fitness values of original and opposite solutions
Step 5: Compare the fitness values and select best values Step 6: Repeat the all solutions in the populations
Step 7: For each candidate Step 8: End for
Step 9: Return the update populations

mappings) starts the optimization process. COBL is used to produce opposite candidate solutions and the best individuals are selected based on a fitness function (e.g., minimizing makespan, response time, and energy consumption). Then, GWO refines the location of solutions in an iterative manner using alpha, beta and delta wolves to direct the search to the optimum resource distribution. In order to maximize the above objective function, the proposed solution uses the GWO. The candidate solutions in GWO are updated according to the locations of three dominant wolves (α, β, δ), which dictate the location of the search. The final position is calculated with the mean of the three best solutions:

$$X(t+1) = \frac{X_\alpha + X_\beta + X_\delta}{3} \quad (33)$$

In order to increase the exploration power, COBL is brought into the GWO system. COBL creates opposite candidate solutions depending on centroid of the population. Centroid is determined as:

$$C = \frac{1}{N}\sum_{i=1}^{N} X_i \quad (34)$$

The opposite solution is well-defined as:

$$X_i^{op} = 2C - X_i \quad (35)$$

Between the original and opposite solutions, the one with better fitness value is selected for the next iteration. This mechanism improves diversity and helps avoid premature convergence. The proposed COBL–GWO model aims to minimize the fitness function while satisfying task allocation constraints, thereby achieving efficient load balancing with improved convergence speed, reduced energy consumption, and better resource utilization in cloud environments.

Table 2 : Properties Settings

Task	
Task Range	10 - 50
Length	1000 - 6000
File size	500
VM	
VM ranges	3 - 5
Speed	225 - 300
Memory	256 - 512
CPU	1 - 5
Bandwidth	1000
VMM	XEN

Algorithm 3: COBL-GWO for load balancing

Step 1: Compute the EET, TTS, and CT
Step 2: Compute the CRU, Load, and capacity of the VM Step 3: Check the load balancing possibility
Step 4: Check the availability of VM Step 5: Initialize the random populations
Step 6: Generate the opposite position of each candidate solution
Step 7: Select the best initial solution from original and opposite solution
Step 8: Evaluate the fitness values
Step 9: Identify the best solutions of X_α, X_β and X_δ
Step 10: For each candidate
Step 10.1: update the control parameters and calculate co-efficient
Step 10.2: Calculate the distance from X_α, X_β and X_δ
Step 10.3: Update position of each candidate
Step 10.4: Apply the COBL to generate population Step 10.4: Select the best original and opposite solution Step 10.5: Re-evaluate fitness values
Step 10.6: Update X_α, X_β and X_δ
Step 10.7: Repeat the steps until meet the convergence Step 11: End for
Step 12: All loads are balanced and tasks are assigned properly

6. Experimental resutls annd analysis

The performance of the proposed COBL-GWO algorithm is measured by comparing it with the well-known algorithms such as IGWO[34], GWO[35], CPSO [36], PSO [37, 38], and GA [39] with the key metrics, including makespan, resource utilization, and execution time. The suggested COBL-GWO load balancing algorithm is deployed and tested with the help of the CloudSim simulative environment. The experimental system is set up on a system with an i7-1235U processor with a frequency of 2.30 GHz, 16 GB RAM and Windows 11 platform (64-bit processor).

The main objective of the work is to reduce makespan and maximize resource utilization in dynamic cloud environments, where it is important to reduce makespan and maximize resource utilization. Table 2 provides a summary of the main features and configuration parameters of the suggested COBL-GWO algorithm. In addition, Table 2 shows the list of non-preemptive independent tasks that are to be considered in the experiment. These are tasks that are assigned to a heterogeneous group of virtual machines in a cloud data center to replicate realistic workload conditions. The heterogeneous VMs used will make sure that the proposed solution is tested in various processing capacities, thus verifying its ability to work in dynamic and complex cloud environments.

6.1 Performance metrics

Three criteria were used to evaluate the suggested LB algorithm's performance in the cloud setting. To measure and evaluate the performance it is measured using the following measures of performance:

I. **Makespan (MT):** MT is the sum of time that will be required in order to schedule a VM. This is mostly used to measure the time-related efficiency of scheduling algorithms.
II. **Execution time (ET):** ET is the exact duration of time that should be taken to accomplish the given tasks on a virtual computer. This will reduce this statistic and therefore the performance of the algorithm will improve.
III. **Resource utilization (RU):** it is a second quantitative statistic that is a parameter dependent statistic. The objective is to improve the efficiency of employing the cloud environment's resources.

6.2 Results analysis

The experimental results presented across multiple performance metrics provide a comprehensive assessment of the proposed COBL-GWO algorithm against five comparative algorithms: IGWO[34], GWO[35], CPSO [36], PSO [37, 38], and GA [39]. The
evaluation is conducted using two configurations—5 and 10 VMs—and across varying task loads from 10 to 50 tasks. VMs are essential components of cloud computing environments' load-balancing systems. To maximize throughput, reduce response times, optimise resource consumption, and not to overload any

single server, LB distributes the incoming network traffic to multiple servers. In cloud computing models, metaheuristic models are significant to LB since they offer good algorithms to maximize the allocation of resources, minimize response time and improve the overall performance of the system.

In the metaheuristic analysis of LB in cloud computing, the performance of the implemented algorithms is evaluated in terms of various criteria, such as resource consumption, reaction time, throughput, scalability, and fault tolerance. Compare the performance of each of the algorithms in distributing the load and allocating resources in the best manner possible under different workload conditions. Compare the LB algorithm with the bandwidth, memory, and CPU requirements of VMs to meet the demands of incoming requests. Determine how much the system is under- or over-utilizing resources and measure the performance of the algorithm in terms of resource optimization. The current paper has concentrated on novel LB techniques like COBL-GWO and the current section covers the capability of COBL-GWO techniques.

Table 3 : Make span value attained for 5 VMs

Tasks	**COBL-GWO**	IGWO	GWO	CPSO	PSO	GA
10	207	271	282	294	308	362
20	306	332	345	406	422	442
30	471	482	512	530	553	566
40	598	618	630	663	691	700
50	710	721	737	754	773	798

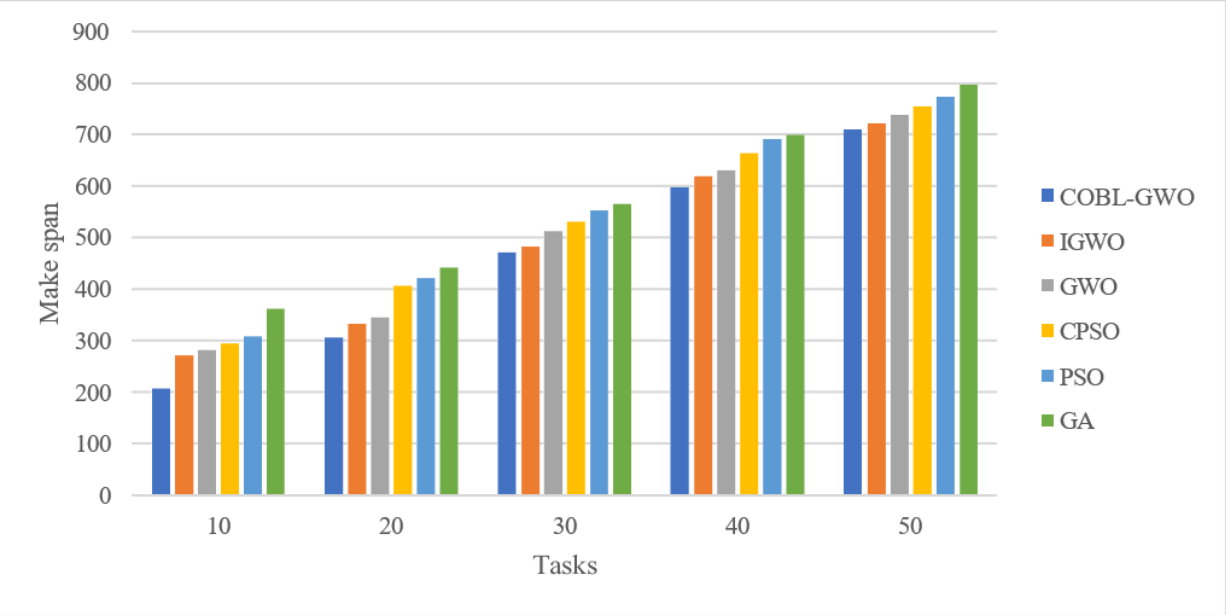

Figure 1 : Make span value obtained for 5 VMs

Table 4 : Make span value attained for 10 VMs

Tasks	**COBL-GWO**	IGWO	GWO	CPSO	PSO	GA
10	105	124	132	152	184	198
20	188	204	225	271	285	302
30	306	312	323	365	372	389
40	398	428	449	463	471	480
50	471	494	508	532	564	596

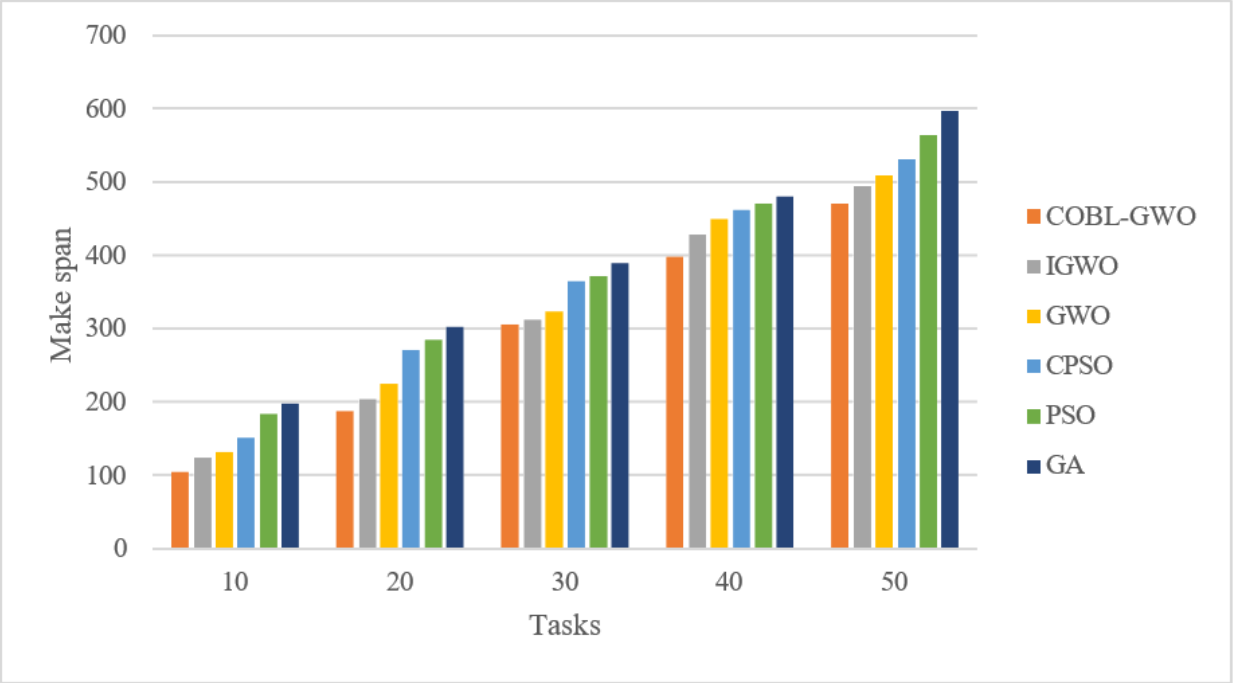

Figure 2 : Make span value attained for 10 VMs

Table 5 : Resource utilization attained for 5 VMs

Tasks	**COBL-GWO**	IGWO	GWO	CPSO	PSO	GA
10	0.74	0.57	0.47	0.41	0.39	0.32
20	0.8	0.63	0.53	0.45	0.39	0.21
30	0.87	0.74	0.62	0.54	0.47	0.38
40	0.9	0.86	0.78	0.7	0.64	0.59
50	0.96	0.88	0.8	0.75	0.7	0.61

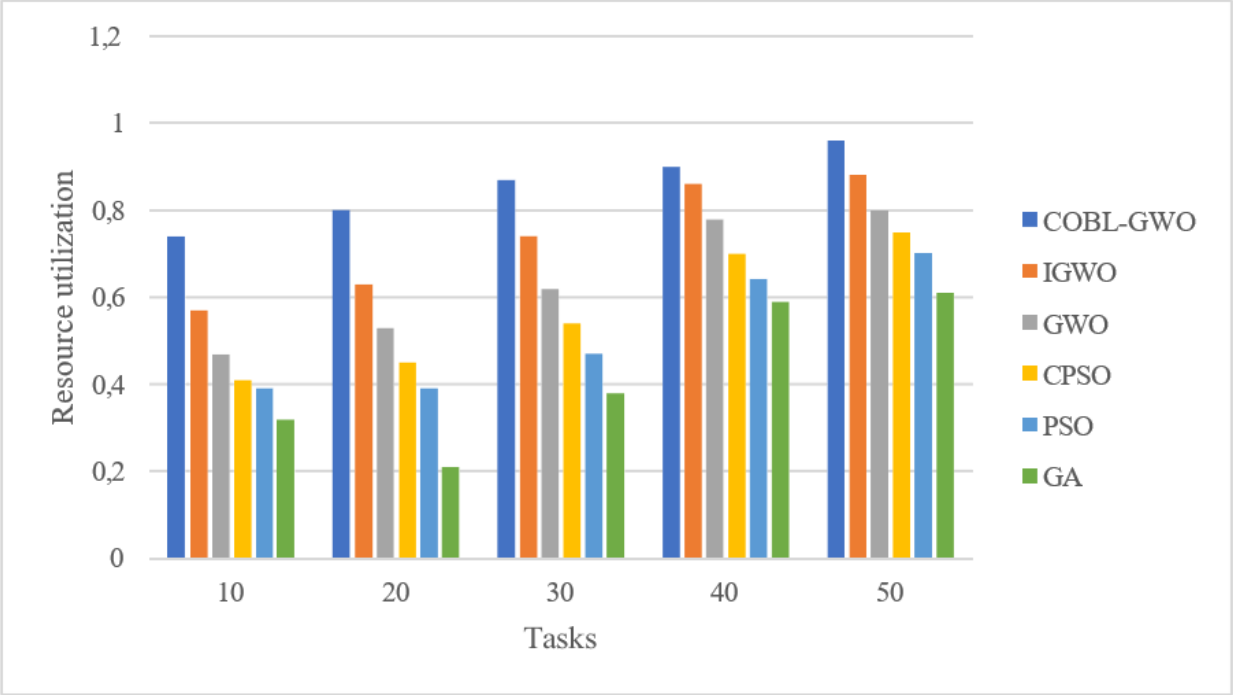

Figure 3 : Resource utilization attained for 5 VMs

Table 6 : Resource utilization attained for 10 VMs

Tasks	**COBL-GWO**	IGWO	GWO	CPSO	PSO	GA
10	0.67	0.53	0.46	0.41	0.37	0.33
20	0.94	0.87	0.7	0.67	0.58	0.45
30	1.08	0.89	0.76	0.68	0.54	0.46
40	1.3	1.02	0.96	0.73	0.64	0.57
50	1.61	1.42	1.1	0.94	0.8	0.73

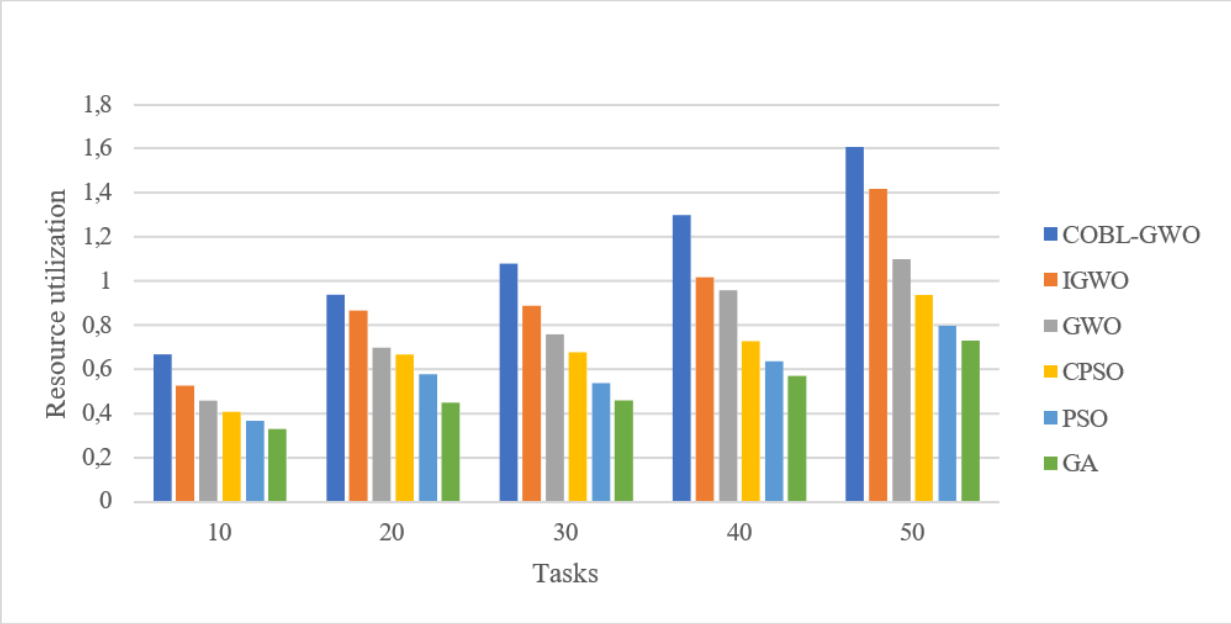

Figure 4 : Resource utilization attained for 10 VMs

Table 7 : Execution time attained for 5 VMs

Tasks	**COBL-GWO**	IGWO	GWO	CPSO	PSO	GA
10	130.73	122.93	103.87	99.08	97.87	96.74
20	163.45	142.65	137.45	130.04	120.98	101.02
30	178.82	165.76	154.87	143.98	135.76	128.09
40	192.76	185.82	179.78	167.89	154.65	147.98
50	228.64	203.46	193.48	187.87	173.52	168.65

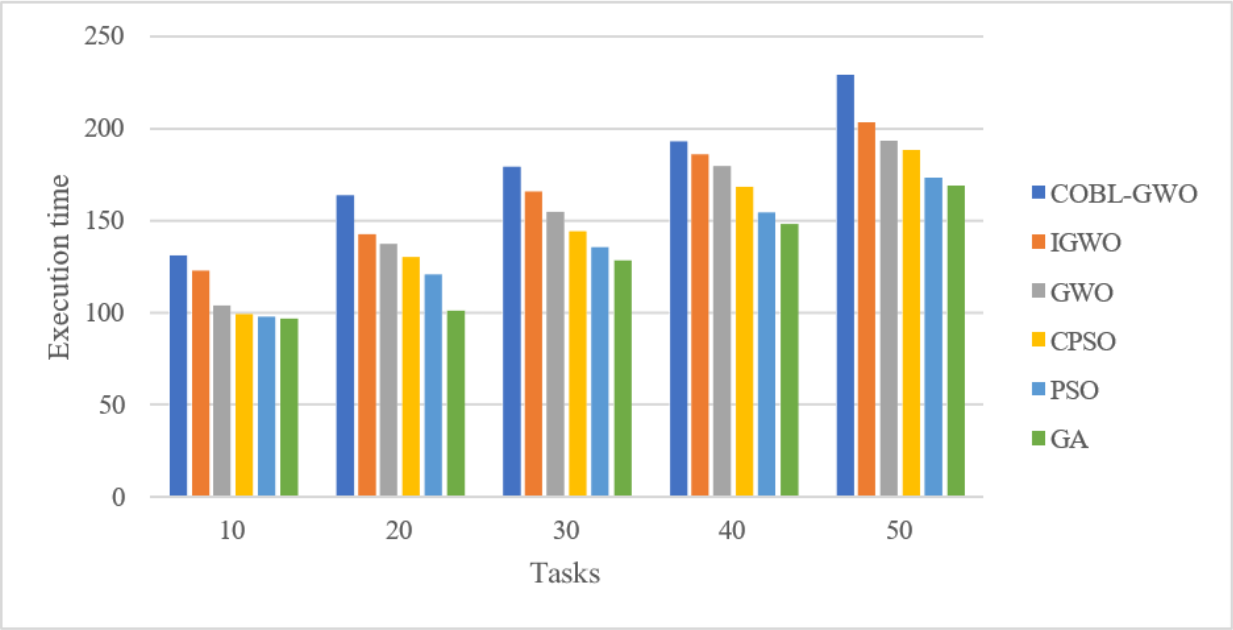

Figure 5 : Execution time attained for 5 VMs

Table 8 : Execution time attained for 10 VMs

Tasks	**COBL-GWO**	IGWO	GWO	CPSO	PSO	GA
10	145.68	133.78	128.46	118.09	113.69	110.72
20	169.46	158.79	147.65	132.58	128.73	117.98
30	330.2	303.65	298.45	286.78	273.03	261.82
40	367.19	359.36	338.76	279.58	267.92	256.73
50	389.42	371.76	364.86	351.43	342.58	328.65

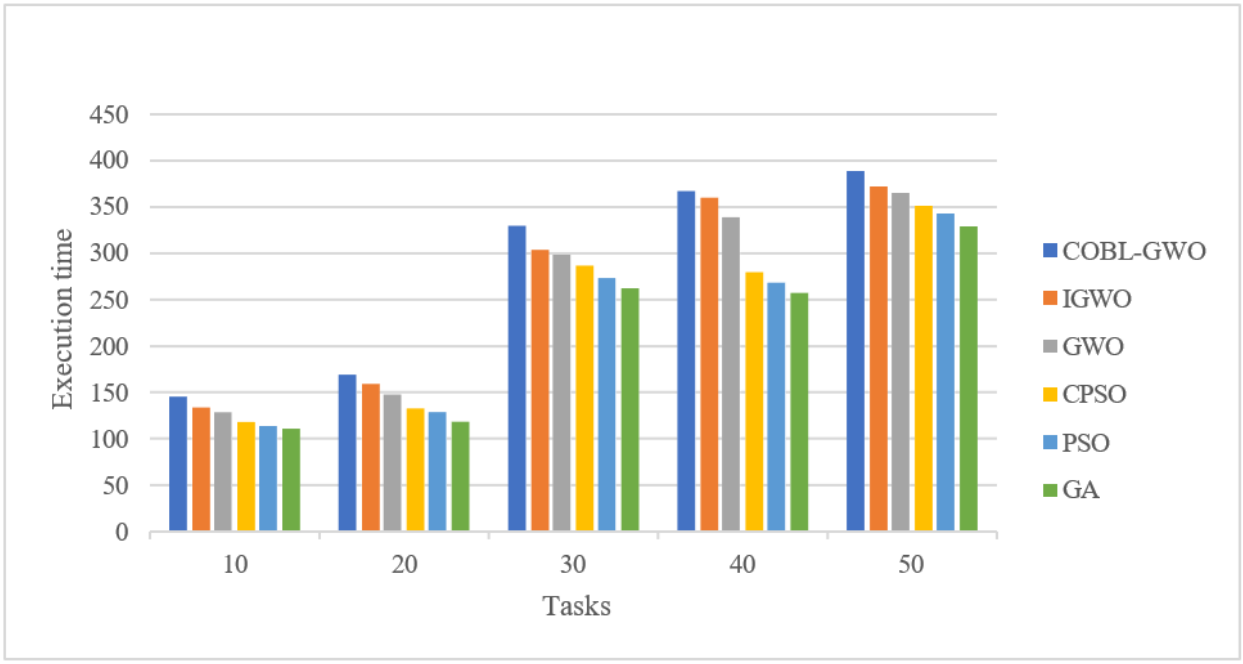

Figure 6 : Execution time attained for 10 VMs

Table 9 : Memory utilization attained for 5 VMs

Tasks	**COBL-GWO**	IGWO	GWO	CPSO	PSO	GA
10	49	45	40	37	35	31
20	46	43	41	38	36	33
30	52	47	45	41	38	35
40	55	49	44	42	39	37
50	59	56	51	48	43	39

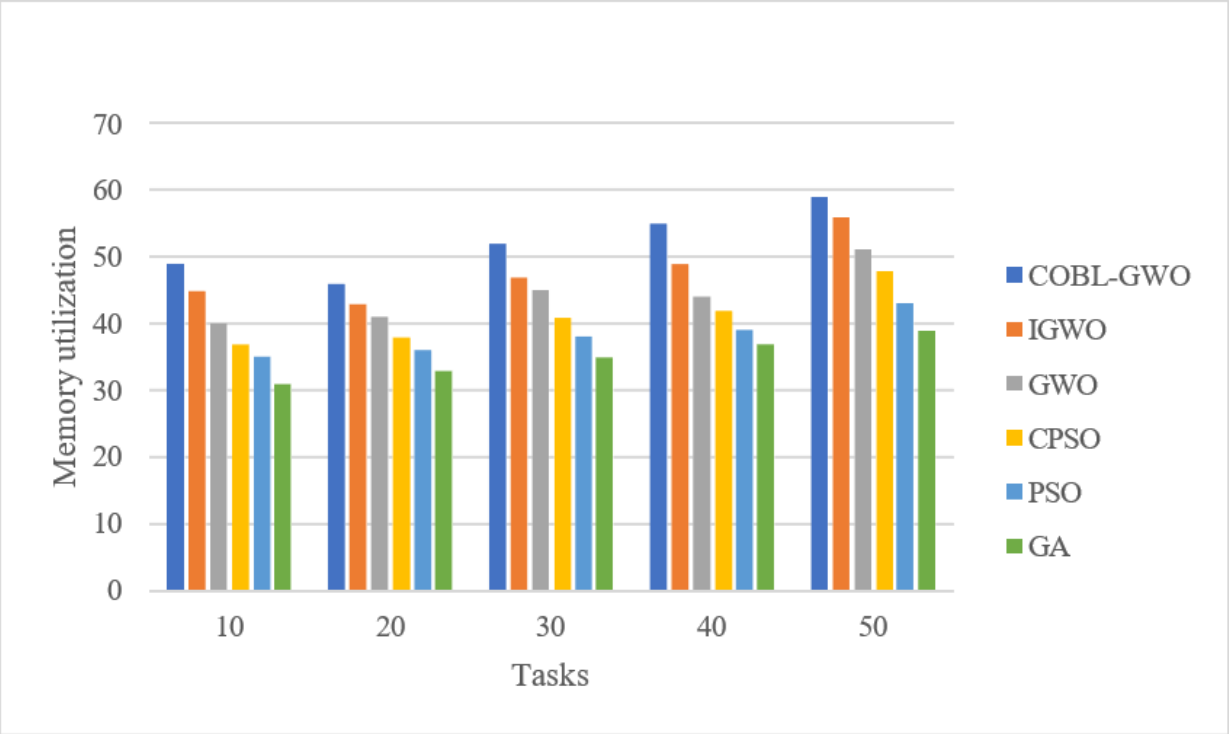

Figure 7 : Memory utilization attained for 5 VMs

Table 10 : Memory utilization attained for 10 VMs

Tasks	**COBL-GWO**	IGWO	GWO	CPSO	PSO	GA
10	55	49	43	40	37	35
20	61	53	49	46	40	38
30	69	63	58	47	45	40
40	72	68	65	58	51	47
50	75	69	61	59	54	50

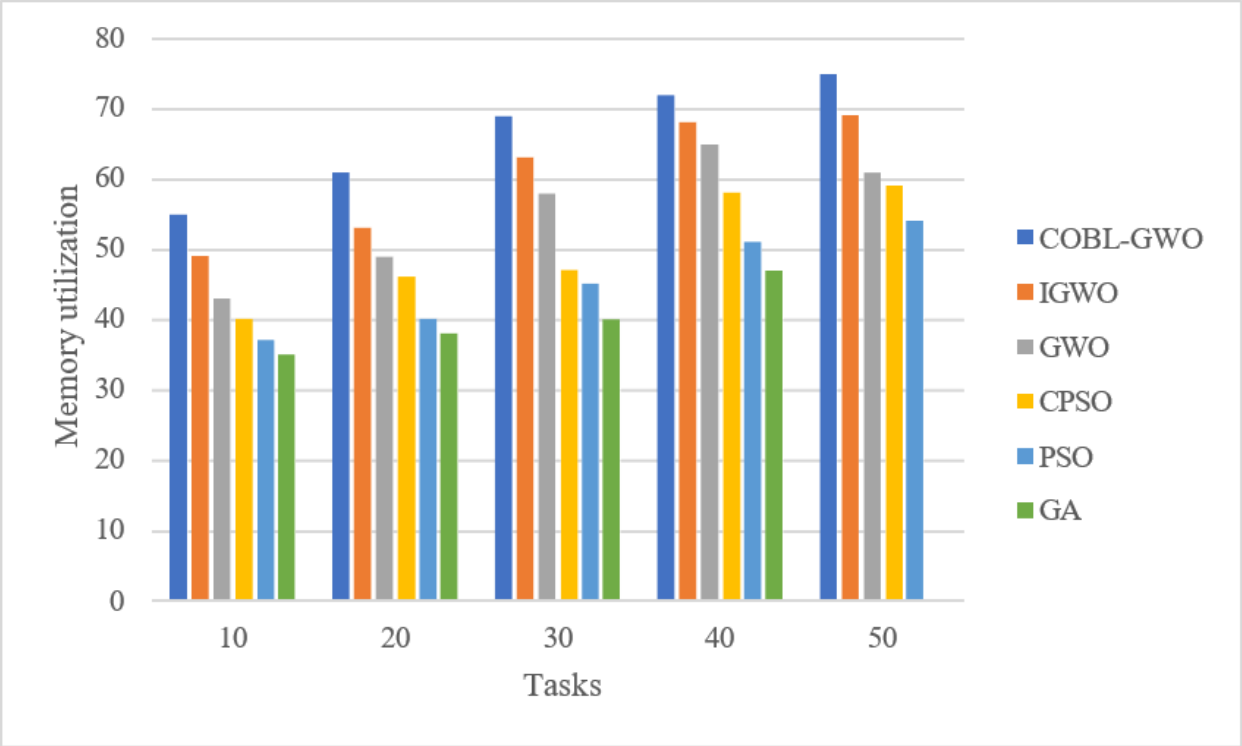

Figure 8 : Memory utilization attained for 10 VMs

COBL-GWO always has the smallest values for make span for both VM configurations, the total time needed to execute all the tasks in the schedule. For example, with 5 VMs, COBL-GWO's make span ranges from 207 to 710 with increasing number of tasks, while that of GA is higher (362 to 798). For COBL-GWO having 5 VMs, the average make span is 458.4, which is 20.08% better than GA with 573.6. There are also similar findings in the 10 VM case; COBL-GWO's average make span is 293.6 while the GA is 393, representing 25.3% improvement. The above results clearly show that COBL-GWO provides better task distribution and efficient scheduling, even under increased task loads, to minimize the total completion time.

In the field of resource utilization (how much of the VM resources are used during its execution), COBL-GWO is once again superior to other algorithms. In the case of 5 VMs, the average utilization of COBL-GWO is 0.854, and the GA is 0.422. COBL-GWO delivers an average utilization of 1.12 with 10 VMs, 1.61 when there are 50 tasks, showing that it can use available resources even in high demands. GA has an average of 0.518, which is lower than the average of 0.675 it had before. This demonstrates that not only is COBL-GWO faster in scheduling tasks, but also it maximizes the use of computational power.

The measure of execution time, however, shows an inverse relationship. GA always shows the least execution times, and COBL-GWO always shows the most execution times in both configurations. For instance, GA takes the average execution time of 128.50 with 5 VMs while COBL-GWO takes 178.08, which is 38.6% more. Likewise, COBL-GWO's average is 280.79 whereas GA's average is 215.58 in the 10 VM setup. This means that there is a clear trade-off: COBL-GWO may result in more complex optimization strategy, thus requiring more computational overhead, but it gives optimized scheduling and resource usage.

The memory utilisation results corroborate the execution time results. COBL-GWO is more memory intensive on average due to the extra processing power needed for optimization mechanism. For 5 VMs, COBL-GWO uses an average of 52.2 MB, compared to GA's 35 MB. The mean of the memory usage of all 10 VMs is shown to be 66.4 MB for COBL-GWO and 42 MB for GA. This means that it costs more but every bit of that is warranted by the tremendous gains in scheduling efficiency and utilization. COBL-GWO outperforms significantly with regard to make span and in terms of resource utilization, especially in an increased number of tasks It takes longer to execute and more memory to use than simpler algorithms such as GA, but the benefits are better management of tasks and improved system efficiency. This makes COBL-GWO very well suited for applications

were making tasks more distributed is more important than minimum execution time, or applications with higher scalability demands. The results of average make span calculated for each algorithm for both 5 VM and 10 VM indicates that COBL-GWO performs better compared with other methods. COBL-GWO is able to reach lower average make span than GA does with 458.4 units while 573.6 units is the average make span of GA with 5 VMs. This is the continuation of the trend, with a COBL-GWO average make span of approximately 293.6 units and GA's average make span of 393 units. The outcomes show that COBL-GWO is very effective at reducing the total completion time of the tasks and can result in faster and more efficient scheduling even with more tasks and virtual machines.

As far as resource utilization is concerned, COBL-GWO once again shows better performance. For 5 VMs, it has an average utilization of 0.854 compared to 0.422 with GA. With this greater utilization, COBL-GWO can better allocate and balance the workload on the resources available, thus minimizing idles and maximizing system efficiency. With 10 VMs, the advantage is even more pronounced, with COBL-GWO achieving an average utilization of 1.12, meaning that it can better handle resources at higher workloads than the other algorithms, which use a lower average of 0.518. Even with improvements in make span and in utilization of resources, execution time results show that there is a definite trade-off involved. The result of the lowest average computation time is obtained by GA, with 128.5 units for 5 VMs and 215.6 units for 10 VMs, whereas the computation time is higher for COBL-GWO with 178.1 units for 5 VMs and

280.8 units for 10 VMs. This is related to the extra computational complexity in the optimization strategy adopted by COBL-GWO, that although increases the quality of the scheduling results, takes more time to compute. The balance of this trade-off reflects the fact that COBL-GWO aims for optimal performance metrics, which are crucial to the efficiency of a cloud, but requires more time to execute.

In the same way, the use of memory shows a similar trend: COBL-GWO uses more memory, on average, than other algorithms. For 5 VMs, COBL-GWO uses about 52.2 MB on average, compared to GA’s 35 MB. The memory requirement increases to 66.4 MB for COBL-GWO and 42 MB for GA for the 10 VM setup. Such high memory utilization is to be expected because of the opposition-based learning and advanced population-based search methods in COBL-GWO. This means more resources are being used, but it's worth it for the many benefits of scheduling efficiency and resource management. COBL-GWO provides a powerful efficient load balancing solution, minimizing make span and maximizing the utilization of resources. The increased computational and memory cost comes with these benefits, though. This trade-off should be taken into account when choosing load balancing algorithms, particularly in scenarios where timely execution and resource limitations are paramount. However, COBL-GWO's ability to gain performance can be a good choice for cloud computing systems where load balancing is important and scalable under dynamic workloads.

CONCLUSION

In this study, a hybrid GWO-COBL load balancing algorithm was presented to enhance the task scheduling and utilization issues in cloud computing systems. Incorporating Centroid Opposition Based Learning increased the exploration capability, solution diversity, convergence speed, and decreased the chance of being stuck in the local optimum. The results of experiments showed that the proposed approach achieved superior performance compared with the other optimization methods like Particle Swarm Optimization (PSO) and Standard GWO (S-GWO) with respect to the objective function makespan, response time, throughput, VM utilization and overall load balancing efficiency. The proposed design was able to support dynamic and heterogeneous cloud workloads, providing a scalable and reliable solution for cloud-based infrastructures. The integration of energy efficiency, fault tolerance, and intelligent deep learning techniques could be explored for future work to further improve the performance of cloud resource management.

References

1. A. Jyoti, A. Yadav, D. Kumar, P. Mamoria, and V. Yadav, "Classification & Technological Analysis of Load Balancing in Cloud Computing," Recent Patents on Engineering, vol. 20, no. 1, p. E18722121358630, 2026.
2. J. Zhou et al., "Comparative analysis of metaheuristic load balancing algorithms for efficient load balancing in cloud computing," Journal of cloud computing, vol. 12, no. 1, pp. 1-21, 2023.
3. K. V. Kumar and A. Rajesh, "Multi-objective load balancing in cloud computing: a meta-heuristic approach," Cybernetics and Systems, vol. 54, no. 8, pp. 1466-1493, 2023.
4. M. Sumathi, S. Raja, A. Jense, A. P. S. Nelsting, and V. Swetha, "Optimal load balancing in cloud computing using hybrid meta-heuristic algorithms," Journal of High Speed Networks, vol. 32, no. 1, pp. 3-17, 2026.

5. M. S. R. Krishna and D. K. Vali, "Meta-RHDC: meta reinforcement learning driven hybrid lyrebird falcon optimization for dynamic load balancing in cloud computing," IEEE Access, vol. 13, pp. 36550-36574, 2025.
6. M. S. Al Reshan et al., "A fast converging and globally optimized approach for load balancing in cloud computing," IEEE Access, vol. 11, pp. 11390-11404, 2023.
7. G. Lokesh and K. Baseer, "A meta-heuristic approach-aided multi-objective strategy with optimal resource allocation via fault tolerant and priority-based scheduling for load balancing in cloud," Wireless Networks, vol. 31, no. 3, pp. 2419-2441, 2025.
8. S. Singhal et al., "Energy efficient load balancing algorithm for cloud computing using rock hyrax optimization," IEEE access, vol. 12, pp. 48737-48749, 2024.
9. K. Geeta and V. Kamakshi Prasad, "Multi-objective cloud load-balancing with hybrid optimization," International Journal of Computers and Applications, vol. 45, no. 10, pp. 611-625, 2023.
10. G. Long, S. Wang, and C. Lv, "QoS-aware resource management in cloud computing based on fuzzy meta-heuristic method," Cluster Computing, vol. 28, no. 4, p. 276, 2025.
11. S. Ghafir, M. A. Alam, F. Siddiqui, and S. Naaz, "Load balancing in cloud computing via intelligent PSO-based feedback controller," Sustainable computing: informatics and systems, vol. 41, p. 100948, 2024.
12. P. S. Priya, S. K. Medishetti, R. Kurupati, R. Brahmini, T. R. Gokul, and S. A. Baji, "PTBO: Energy and Makespan-Aware Scheduling in Cloud Computing Using Meta-Heuristic Algorithm," in 2025 6th International Conference on Data Intelligence and Cognitive Informatics (ICDICI), 2025: IEEE, pp. 562-569.
13. M. Sumathi, N. Vijayaraj, S. P. Raja, and M. Rajkamal, "HHO-ACO hybridized load balancing technique in cloud computing," International Journal of Information Technology, vol. 15, no. 3, pp. 1357-1365, 2023.
14. P. J. Assudani and P. Balakrishnan, "An efficient approach for load balancing of VMs in cloud environment," Applied Nanoscience, vol. 13, no. 2, pp. 1313-1326, 2023.
15. S. Singhal, N. Gupta, P. Berwal, Q. N. Naveed, A. Lasisi, and A. W. Wodajo, "Energy efficient resource allocation in cloud environment using metaheuristic algorithm," Ieee Access, vol. 11, pp. 126135-126146, 2023.
18.
16. M. Jesi, A. Appathurai, M. NARAYANAPERUMAL, and A. Kumar, "Load balancing in cloud computing via mayfly optimization algorithm," Revue Roumaine Des Sciences Techniques—Série Électrotechnique Et Énergétique, vol. 69, no. 1, pp. 79-84, 2024.
17. K. J. Rajashekar, Channakrishnaraju, P. C. Gowda, and A. B. Jayachandra, "SCEHO-IPSO: a nature-inspired meta heuristic optimization for task-scheduling policy in cloud computing," Applied Sciences, vol. 13, no. 19, p. 10850, 2023.
18. R. Zhanuzak, M. A. Ala'Anzy, M. Othman, and Algarni, "Optimizing cloud computing performance with an enhanced dynamic load balancing algorithm for superior task allocation," IEEE Access, vol. 12, pp. 183117-183132, 2024.
19. V. Hayyolalam and Ö. Özkasap, "CBWO: a novel multi-objective load balancing technique for cloud computing," Future Generation Computer Systems, vol. 164, p. 107561, 2025.
20. T. A. Murshedi et al., "Optimizing Cloud Computing: A Holistic Strategy for Efficient and Sustainable Operation through Ant Colony Load Balancing and Resource Optimization," International Journal of Intelligent Engineering & Systems, vol. 17, no. 5, 2024.
21. K. Ramya and S. Ayothi, "Hybrid dingo and whale optimization algorithm-based optimal load balancing for cloud computing environment," Transactions on Emerging Telecommunications Technologies, vol. 34, no. 5, p. e4760, 2023.
22. F. Kaplan and A. Babalik, "Performance analysis of cloud computing task scheduling using metaheuristic algorithms in DDoS and normal environments," Electronics, vol. 14, no. 10, p. 1988, 2025.
23. J. P. Gabhane, S. Pathak, and N. M. Thakare, "A novel hybrid multi-resource load balancing approach using ant colony optimization with Tabu search for cloud computing," Innovations in Systems and Software Engineering, vol. 19, no. 1, pp. 81-90, 2023.
24. U. K. Lilhore et al., "A multi-objective approach to load balancing in cloud environments integrating ACO and WWO techniques," Scientific Reports, vol. 15, no. 1, p. 12036, 2025.
25. P. Neelakantan and N. S. Yadav, "An optimized load balancing strategy for an enhancement of cloud computing environment," Wireless Personal Communications, vol. 131, no. 3, pp. 1745-1765, 2023.
26. K. Nivitha, P. Pabitha, and R. Praveen, "CBBM-WARM: A workload-aware meta-heuristic for resource management in cloud computing," China Communications, vol. 22, no. 6, pp. 255-275, 2025.
27. A. Choudhary and R. Rajak, "A novel strategy for deterministic workflow scheduling with load balancing using modified min-min heuristic in cloud computing environment," Cluster Computing, vol. 27, no. 5, pp. 6985-7006, 2024.
28. L. Du and Q. Wang, "Metaheuristic Optimization for Dynamic Task Scheduling in Cloud Computing Environments," International Journal of Advanced Computer Science & Applications, vol. 15, no. 7, 2024.
29. P. Geetha, S. Vivekanandan, R. Yogitha, and M. Jeyalakshmi, "Optimal load balancing in cloud: Introduction to hybrid optimization algorithm," Expert Systems with Applications, vol. 237, p. 121450, 2024.
30. A. Boroumand, M. Hosseini Shirvani, and H. Motameni, "A heuristic task scheduling algorithm in cloud computing environment: an overall cost minimization approach," Cluster Computing, vol. 28, no. 2, p. 137, 2025.

31. N. Elsakaan and K. Amroun, "A novel multi-level hybrid load balancing and tasks scheduling algorithm for cloud computing environment: N. Elsakaan, K. Amroun," The Journal of Supercomputing, vol. 80, no. 9, pp. 13434-13474, 2024.
32. S. Mirjalili, S. M. Mirjalili, and A. Lewis, "Grey wolf optimizer," Advances in engineering software, vol. 69, pp. 46-61, 2014.
33. P. Rajesh and K. Maharajan, "An Efficient Load Balancing Scheme in Cloud Computing using Gray Wolf Optimization with Centroid Opposition-Based Learning Approaches," in 2025 9th International Conference on Inventive Systems and Control (ICISC), 2025: IEEE, pp. 1811-1818.
34. B. B. Aburinya, M. I. Daabo, and C. I. Nakpih, "A proposed enhanced dynamic grey wolf optimization algorithm for cloud load balancing," Journal of Electrical Systems and Information Technology, vol. 13, no. 1, p. 27, 2026.
35. S. Sefati, M. Mousavinasab, and R. Zareh Farkhady, "Load balancing in cloud computing environment using the Grey wolf optimization algorithm based on the reliability: performance evaluation," The Journal of Supercomputing, vol. 78, no. 1, pp. 18-42, 2022.
36. R. Vadivel and T. Sudalaimuthu, "Cauchy particle swarm optimization (CPSO) based migrations of tasks in a virtual machine," Wireless Personal Communications, vol. 127, no. 3, pp. 2229-2246, 2022.
37. F. Ramezani, J. Lu, and F. K. Hussain, "Task-based system load balancing in cloud computing using particle swarm optimization," International journal of parallel programming, vol. 42, no. 5, pp. 739-754, 2014.
38. B. Mondal, "Optimization techniques for big data: A VM load balancing in cloud computing using particle swarm optimization," in Mathematical Modeling for Big Data Analytics: Elsevier, 2026,pp. 143-156.
39. K. Dasgupta, B. Mandal, P. Dutta, J. K. Mandal, and S. Dam, "A genetic algorithm (ga) based load balancing strategy for cloud computing," Procedia Technology, vol. 10, pp. 340-347, 2013.